ARMY RESEARCH LABORATORY

# The Communications Maintenance and Optimization Problem in Robotic Multi-hop Networks: Assumptions, Models, and Algorithms

by Cem Karan

ARL-TR-6342 February 2013

## NOTICES

### Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

# Army Research Laboratory

Adelphi, MD, 20783-1197

**ARL-TR-6342** **February 2013**


# The Communications Maintenance and Optimization Problem in Robotic Multi-hop Networks: Assumptions, Models, and Algorithms


**Cem Karan**

**Computational and Information Sciences Directorate, ARL**

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED (From - To) |
|---|---|---|
| February 2013 | Final | August 2011-September 2012 |

**4. TITLE AND SUBTITLE**
The Communications Maintenance and Optimization Problem in Robotic Multi-hop Networks: Assumptions, Models, and Algorithms

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**
Cem F. Karan

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
U.S. Army Research Laboratory
ATTN: RDRL-CII-A
2800 Powder Mill Rd.
Adelphi, MD 20783-1197

**8. PERFORMING ORGANIZATION REPORT NUMBER**
ARL-TR-6342

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

**10. SPONSOR/MONITOR'S ACRONYM(S)**

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
Approved for public release; distribution is unlimited.

**13. SUPPLEMENTARY NOTES**
Author's email: cem.f.karan.civ@mail.mil

**14. ABSTRACT**
An important problem in robotics is the Communications Maintenance and Optimization Problem (CMOP). In this problem, we have a set of robots that are communicating with one another across a multi-hop wireless network. The primary constraint is that the robots must maintain communications between all pairs of robots, ideally in an optimal manner, while performing their assigned tasks. In this report, I describe a set of constraints and metrics for roboticists to use while trying to solve this problem. I define the Useful Model, which is only able to model a subset of those constraints and metrics, and analyze how it limits what controllers can be created using the information it contains. Next, I describe two data models that contain subsets of the information in the model that try to solve the communications problem, showing how the limitations of the model hinder the ability of the controllers to solve the problem. Finally, I describe several ways to extend the model to include more information, which may lead to better controllers in the future.

**15. SUBJECT TERMS**
robotics, communications, control, C2

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Cem Karan |
| Unclassified | Unclassified | Unclassified | UU | 40 | 19b. TELEPHONE NUMBER (*Include area code*) 301-394-0667 |

**Standard Form 298 (Rev. 8/98)**
**Prescribed by ANSI Std. Z39.18**

# Contents

## List of Figures

# 1. Introduction

This section is intended to give the reader an intuitive understanding of the Communications Maintenance and Optimization Problem (CMOP), and some of the factors that affect it. Because the goal is more of an intuitive understanding than a mathematical one, this section is intentionally light on mathematical detail. The information in this section should be used as a guide to some of the important problems that designers face when trying to solve the CMOP, rather than as a rigorous treatment of the subject.

## 1.1 The Communications Maintenance and Optimization Problem (CMOP)

One area of robotics that is rapidly gaining importance is communications. In the past, the assumption has been that if robots need to communicate, then they will always be able to do so across a network that already exists and is sufficiently capable of meeting all of the robot's needs. This is not always true; in some cases, no network exists and robots must form their own ad-hoc wireless networks. In these cases, a team of robots may need to constrain their actions to ensure that they always meet minimum networking constraints and requirements.

Moreover, even if we meet the minimum requirements, we often have an idea of what we consider to be an optimal network. If a team of robots can communicate across an optimal network, then they may be able to perform their tasks with greater efficiency than otherwise possible.

This suggests that there are two factors involved in optimizing a communications network: constraints and optimality. From these observations, we can define the CMOP, but first here are some helper definitions:

- Let $R$ be a set of robots, where for each robot $r \in R$, $r$ has the following associated with it:
  - A set of variables (metrics) that it can measure, $M_r$.
  - A configuration space $C_r$ that describes all poses and configurations that robot $r$ can attain.
- Let $M = \bigcap_{r \in R} M_r$.
- Let $M_c \subseteq M$ be a subset of metrics that can be used to measure communications quality in some manner. Let $N_c$ be the space spanning $M_c$.
- Let $\mathcal{A}_c$ be a set of non-overlapping bounded shapes embedded in $N_c$ such that each shape $A_c \in \mathcal{A}_c$ is a region of values that are considered acceptable network values.
- Let $O$ be a measure of optimality over vectors $\vec{m}$, where the elements of $\vec{m}$ are variables in $M_c$. In short, $\vec{m}$ is a point in $N_c$, and $O$ describes how "good" the point is.

- Define two robots $r_1, r_2 \in R$ as being "connected" iff the measure of their connection over the space $N_c$ is inside of some $A_c \in \mathcal{A}_c$.[1]

At this point, we can define the CMOP.

**Definition 1**
*Constraints -*

*Given a configuration space $C_r$ for each robot $r \in R$, and a set of acceptable communications shapes $\mathcal{A}_c$, ensure that the trajectories of the robots are such that $\forall r_1, r_2 \in R$, $r_1$ and $r_2$ are always connected.*

*Optimality -*

*Given an optimality measure $O$ find trajectories for each robot $r$ in $C_r$ that seeks an optimal point $o_{r,t} \in N_c$ that is optimal by $O$'s measure, without violating the constraint above*[2]*.*

A controller that attempts to solve the CMOP must satisfy the constraints and may seek an optimum as defined in Definition 1. As noted in section 1.2, task completion may be considered either a constraint or a metric to be optimized.

Definition 1 describes the communications problem, but also exposes many questions. For example, what constraints should we consider? How do we measure optimality? Will the constraints and optimality measures remain the same over the lifetime of the network? Can we measure what we want to optimize? Do the constraint variables and the optimization variables exist in the same subspace? These and other questions are considered in the rest of this section.

### 1.2 Metrics and Constraints

To solve the CMOP, designers need information, which means metrics. The metrics chosen can then be optimized or they can be used as constraints that must not be violated. As an example of types of metrics, consider the following list:

[1] Since the networks are permitted to be multihop networks, there is a path leading from $r_1$ to $r_2$; it does not imply that $r_1$ must be directly connected to $r_2$.

[2] $o_{r,t}$ may change as the robot moves in configuration space. This is why the optimal point is also a function of time $t$.

- Connectivity
- Throughput
- Latency
- Reliability
- Power consumption
- Task completion (may be complete/incomplete or a percentage)

This list has a number of problems, but I address just one in this section, leaving the other problems for other sections.

This list is incomplete. Network users will always develop new requirements that are not being measured and demand solutions that can solve the problem using the new metrics. The problem is that not all measures can be reduced to the same type. As an example, consider task completion. It can be modeled as a Boolean or it can be modeled as a real number in the interval $[0, 1]$[3]. Certain types of controllers can handle Booleans well, while others (like gradient descent) can handle a real interval well. This means that designers cannot simply plug in a new metric and expect their controller to work. They have to know the type that the metric reduces to or design a new controller, or they have to choose which metrics their controller can handle and explicitly state that it cannot work with other metrics.

### 1.3 Assumptions and Abstractions

Because both the configuration space $C_r$ and the communications quality space $N_c$ may be both large and have high dimensionality, algorithm designers must often make simplifications to make the `CMOP` tractable. These simplifications fall broadly into assumptions and abstractions about the problem. These include abstractions about how to model data, robot behavior, etc. They also make assumptions about the world. Since these are often interrelated, we have chosen not to try to separate them completely, but instead list what is often done[4].

Note that the simplifications that are listed here are simplifications that the author has noticed or deduced from reading a variety of papers; other researchers may notice a different set of 'common' simplifications either because they read a different set of papers than the author, or because they have deduced a different set than the author has.

[3] It can also be modeled as a percentage, a fraction, or may other ways. This is just one way.

[4] I would like to continue expanding this section. If any readers are aware of other commonly made abstractions, please contact me so that may add them to the list.

### 1.3.1 Assumptions About Robots

Robots are failure prone. They may not fail immediately, but given enough time, they will fail. The problem is that not all failures are created equal. As an example, consider the National Aeronautics and Space Administration's (NASA) Spirit rover on Mars. One of its wheels failed in March, 2006 (*1*), but despite this, Spirit continued to operate until March 22, 2010 (*2*). In this case, Spirit continued to be useful for 4 years after its initial failure. Contrast this with a bomb disposal robot, which may be completely destroyed in during its mission. The variety of failure modes are hard to fully model, which leads designers to make one of the following assumptions about robots under their control:

- Robots never fail.
- When robots fail, it is a complete and permanent death, with no possibility of recovery.
- Robots may fail temporarily and recover later on. It is impossible to decide if a robot has permanently failed or if the failure is temporary.

All of these assumptions are wrong. In the last case, it is possible to decide that if a robot is sufficiently damaged, then it can never rejoin the network (e.g., a bomb disposal robot that was observed being destroyed).

Roboticists often assumes that robots have enough power and energy to fulfill any commands given them. The problem with this assumption is that a robot may be operated such that it consumes all available power and finally fails due to a lack of energy. An example of this is when an aerial vehicle like a quadrotor is used as a communications relay station. Since quadrotors are like helicopters, they require a great deal of power to remain aloft. This means that if they are driven to a point in space that is optimal for network communications, and that point requires them to fly, they are likely to run out of energy quickly. If energy and power are made a part of the optimization problem, then the quadrotors may be driven to land somewhere, which may be suboptimal in terms of network metrics, but is much better in terms of energy conservation.

Finally, roboticists often assumes that all robots in a group are similar unless otherwise stated. This is not always the case, especially as old robots are upgraded or replaced by newer versions with differing capabilities. Sometimes these changes have an effect on the problem, while other times they do not. Without analysis of the exact changes, it is impossible to determine the effects of the changes. In my opinion, it is always safest to assume that a set of robots is heterogeneous.

### 1.3.2 Assumptions About Sensing and Measurements

In order for a robot in the real world to make decisions (including ones that affect finding solutions to the CMOP), robots need to take measurements. Taking measurements means having sensors. However, sensors generally cannot measure every aspect of the environment nor can they be guaranteed to be 100% accurate. In fact, sometimes they cannot measure the metric that we really want at all. All of these problems lead to the following assumptions that roboticists commonly make:

- **All measurements are precise and accurate.** This is not a universal assumption some researchers assume that measurements are probabilistically correct and they assign an estimate of correctness to each measurement.
- **The environment is either fully mapped, static, or both.** While there are many papers that assume that the environment is unknown, I have not seen any papers where the environment is assumed to be dynamic. However, I question the usefulness of papers on search-and-rescue or search-and-destroy missions that assume the environment is static or quasi-static.
- **Proxy measures are accurate gauges of what really needs to be measured.** A network can be considered "good" if it meets the needs of its users at all times. However, even if this can be measured objectively by the robots, it may take too long for the robots to measure it and still react in time. For this reason, measures like Received Signal Strength Indicators (RSSIs), which are easy and quick to measure, may be substituted for the measure of interest. These proxies are then assumed to correlate well with the measures of interest and are what the system optimizes for.

The last point is worth expanding on. If the proxy measures do not correlate well, or worse, only correlate well over the limited set of tests that the roboticist made, then the claimed solution may not truly solve the CMOP. If the roboticist is lucky, the proxy is fairly close to the actual measure of interest and any solutions will be close to the true optimum. If the roboticist is unlucky, then the solution found will be suboptimal. If the roboticist is truly unlucky, then using the proxy will lead to a solution that violates the constraints of the system. Only by using the actual measure of interest can the CMOP be truly solved.

### 1.3.3 Assumptions About Communication Models

One of the simplest abstractions that designers make is to assume that a single value is sufficient to fully specify the performance of the network. This can be seen when designers assume that the network is "fast enough" and then look at packet reliability figures, or when they assume that a certain level of connectivity ($k$-connectivity) means that the network is "good," and then optimize

a different parameter within the connectivity constraint. This immediately discounts the possibility of a Pareto optimal frontier, which may be more common than many researchers expect. As pointed out in reference 5 and section 1.2, different messages may have very different networking needs, which in aggregate may create a Pareto optimal frontier[5]. The problem with ignoring this frontier is that different portions of the frontier may change at different rates and by different amounts over time. By choosing an operating point along a slowly changing portion of the frontier, designers increase the likelihood that the robots in the network will be able to keep up with the changes. To the best of my knowledge though, this is not currently done in any algorithm or actual system.

Another common assumption is that the graph of network connectivity can be modeled by an undirected graph. This immediately implies that for any pair of nodes $A$ and $B$, all measurements that $A$ makes in relation to $B$ are immediately reflected. Unfortunately, even for groups of identical robots, this is not necessarily true, because the environment may be anisotropic in relation to the measure in use, making the undirected graph assumption immediately false. As an example, consider two robots that are attempting to communicate, while one of them is being jammed by a directional jammer (figure 1). In this case, the environment has conspired to ensure that an undirected graph model of communications will always be false; either it can model both robots as being able to communicate or neither one being able to communicate. With a directed graph model, the jammed robot can transmit its position and the noise level it is measuring, possibly permitting other robots to come close to it to improve communications.

Combining the above assumptions, designers often simplify network graphs to one of the following:

- A communications graph that is undirected and has only one parameter of how "good" a link is.
- A communications graph that is directed, but with only one parameter of how "good" a link is between pairs of nodes.

From the previous comments, it should be obvious that both assumptions hide the truth.

### 1.3.4 Assumptions About Planning

Planning where to move individual robots (or even groups of robots) is essentially an optimization problem. The difficulty is that many of the favored methods of finding an optimal

[5]My assumption is that the rate that messages are generated is sufficiently high that it is impossible for a robot to optimize its pose and radio parameters for each message individually. This necessitates making tradeoffs, which is where the Pareto optimal frontier comes from.

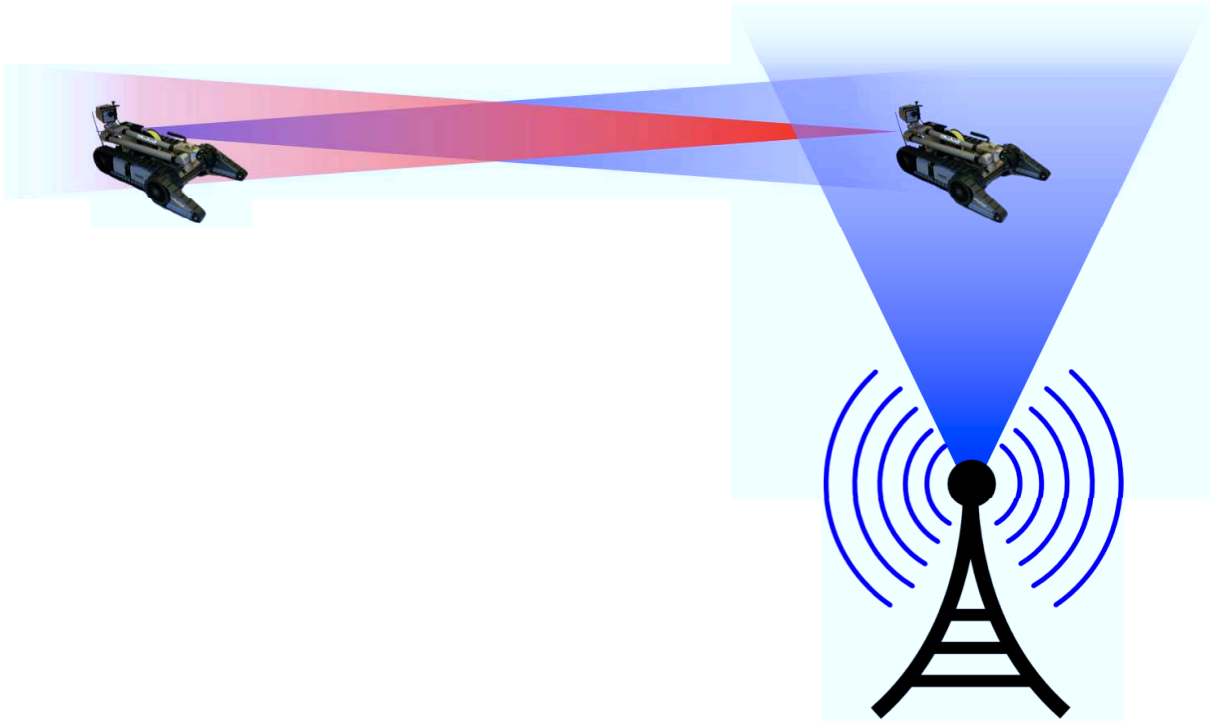


Figure 1. An anisotropic communications example. One robot is being jammed by a directional jammer, causing communications to be unidirectional, even with identical robots.

solution impose their own constraints. Some examples of assumptions that are made when designing planners are the following[6]:

- The CMOP is already a convex problem or can be converted into a convex problem.
- All messages require the same latency/robustness/etc. and this is constant over the lifetime of the network.
- Central planning is possible and better than distributed planning (or vice-versa).
- Planning of any kind is impossible and robots need to react to changes in the network using simple tactics.

Planners cannot assume that the CMOP is generally convex because free space is generally not convex. If a robot needs to move from where it currently is to another point in space in order to improve communications, it may have to move along a trajectory that makes communications worse before it reaches the optimal point. Designers fudge around this by either partitioning free space into a set of convex problems (e.g., convex cell decomposition) or requiring users of the

[6]Some of these are mutually exclusive assumptions.

algorithm to define functions that convert all measurements into a convex space. Unfortunately, to the best of my knowledge, there are no known methods of converting arbitrary non-convex problems into convex ones that guarantee an optimal solution for the original problem. Thus, assuming the `CMOP` can be modeled or converted into a convex problem is generally incorrect.

In addition, not all messages have the same constraints. That means that planners must either adapt to changing network needs or force the robots into positions that guarantee that the network is overengineered. Since knowing about future needs is often difficult, if not impossible, designers usually choose to overengineer the network, making guarantees about what the network can provide as minimal as possible. That said, if there is a high expectation that a robot will operate in some mode for a time and then switch modes, it may make sense to plan the trajectories of the robots to support those planned changes. This could improve the quality of the network over time and may even allow certain operations to occur that would otherwise be impossible.

Some papers attempt to use distributed planners, while others assume that there is enough bandwidth that all nodes can relate their state information to some central planner, which then plans all moves. The problem with central planners is that if a node loses contact with the planner, then it cannot recover (unless there is a prearranged backup plan). The problem with distributed planners is they generally have only incomplete information about the world and must make assumptions (e.g., the problem is convex), which may be invalid. Good arguments can be made for either centralized or decentralized planners, which is why deciding which is best is still an active area of research within the robotics community.

In other papers, the designer does not even bother trying to predict where to place robots, but instead simply reacts to the current state of the network. The result is a purely reactive strategy, such as the one outlined in reference 4. These strategies tend to be quite simple, which makes them easier to implement. Unfortunately, they are unable to optimize a network for future needs, so even when it is possible to determine what networking needs there will be in the future, that information cannot be used.

### 1.3.5 The CAP Theorem

Finally, anyone that works with networks faces the results of the CAP theorem (see reference 6). In a network, we want consistency (all nodes see the same data at the same time), availability (all communications receive a response), and partition-tolerance (the system continues to work even if the network is partitioned).

Unfortunately, the CAP theorem proves that we cannot guarantee all three properties at the same time in any network; at most, we can guarantee a network has any two of the properties. The literature shows that most roboticists who are working on the CMOP are at least intuitively aware that having all three properties is difficult, but few seem to be aware that it is impossible. The result is that roboticists often either make a set of assumptions that guarantees at least one of the properties is true or they relax the primary constraint of Definition 1 so that the problem is only probabilistically solved. The papers analyzed in this report illustrate both of these approaches.

---

## 2. A Useful Model for the CMOP

---

"All models are wrong, but some are useful." (George E. P. Box)

In section 1, we outlined different constraints and abstractions that have been commonly used by various researchers while trying to solve the CMOP. As stated there, the list of problems is incomplete. This will necessarily limit the number and type of data models it is possible to create. In addition, designers will choose which subproblems are important to emphasize, which will lead to data models that have different types of information, as well as different quantities of information. In this section, we define one particular model that we find useful. This model (henceforth known as the Useful Model) is a superset of models used in the papers analyzed in this report. It can model a moderately broad amount of information about the network, but as discussed at the end of this section, because of the design choices in what it models, the abilities of the controllers that can use it are limited. In the following sections, where different solutions to the CMOP are explored, we also explore how these limitations directly affect those solutions.

### 2.1 Model Definition

The model we use is summarized in figure 2. In the figure, there are several variable types defined:

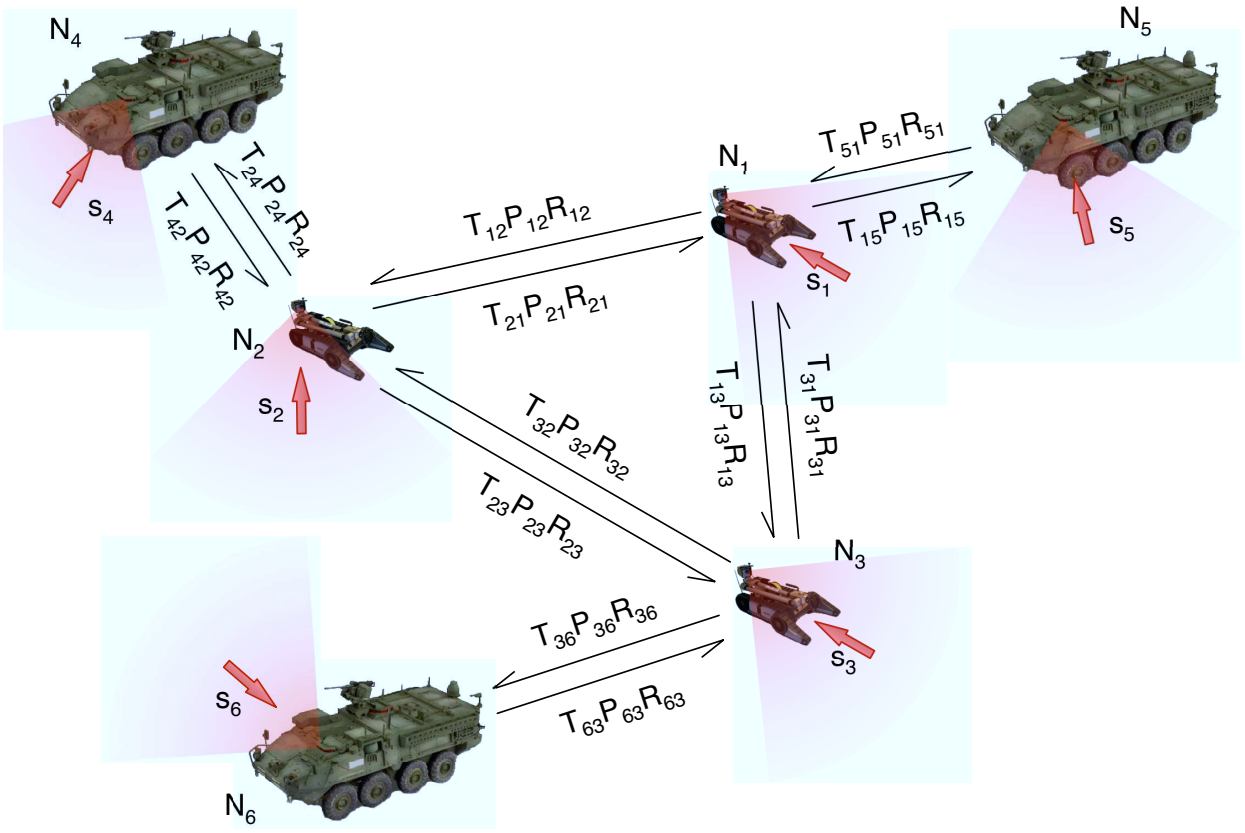



Figure 2. The `Useful Model` generalizes the one in reference 3. All members are peers, and it is entirely possible for $N_4$, $N_5$, and $N_6$ to communicate with each other and with more robots, but due to space limitations, these links were not drawn.

- $\mathbf{N_i}$ - These are the nodes in the network. All nodes are equivalent and are peers. In figure 2, two different types are shown to make it easier to compare and contrast the model with figure 4 in section 4.
- $\mathbf{s_i}$ - Each node in the network can generate messages that we wish to transmit on the network. Node $N_i$ generates messages at a normalized average rate of $s_i$. The normalized range is over the real interval $[0, 1]$. These messages may be sensor measurements (shown in figure 2), commands, or other types of network traffic.
- $\mathbf{T_{ij}}$ - This is the probability that node $N_i$ will choose to transmit a message to its neighbor $N_j$; this is something that node $N_i$ has complete control over. This value is always in the real interval $[0, 1]$. Note that $T_{ij}$ does not necessarily equal $T_{ji}$.
- $\mathbf{P_{ij}}$ - This is the probability that a message will be successfully transmitted (not be lost due to noise or other problems not controlled by the nodes) from node $N_i$ to its immediate neighbor

$N_j$. The value is always in the real interval $[0, 1]$. No node has any control over this value. Note that $P_{ij}$ does not necessarily equal $P_{ji}$.

- $\mathbf{R_{ij}}$ - This is the normalized maximum rate that messages can be sent from node $N_i$ to node $N_j$. This value is analogous to the maximum bit rate of a link, but in terms of messages, with values normalized across the network. The value is normalized to the real interval $[0, 1]$.
- $\mathbf{x_i}$ - This is not shown in figure 2 due to a lack of space, but it is the pose of robot $N_i$. This is always a vector, of whatever size is necessary to ensure that it can completely encode the pose information of robot $N_i$.

The network model assumes that the smallest unit of communications is a message or packet, and it makes the following assumptions about them:

- Messages are indivisible and are passed along or lost in their entirety.
- Messages have a maximum length in octets. Messages may be shorter than this maximum, but never longer.
- Each peer can generate, consume, pass along, or drop messages.
- All messages are considered to be equally important; messages cannot signal their individual reliability, latency, or other metrics to the network.

Since the rate that messages are received at and are transmitted at can differ substantially, each peer has its own in-order, finite length incoming message queue. Messages are drained from the queue to transmit to recipients[7]. To prevent the queue from being completely filled up, the queue uses a randomized leaky bucket strategy[8]. The strategy uses the variables $l$, which is the current number of messages in the queue, and $m$, which is the maximum number of messages permitted in the queue. Each time a message is to be added to the queue, one message is chosen uniformly at random from the set $\{\text{incoming message}\} \cup \{\text{messages in queue}\}$, and then with probability $l/m$ is discarded. When the queue is empty or nearly so, it is unlikely that messages will be dropped. As the queue fills up, it is more likely that a given message will be dropped.

[7] Because this is an incoming message queue, even messages that are destined for the current node are processed through the queue. Thus, these messages are affected by the leaky bucket strategy.

[8] See reference 7 for more details on leaky bucket strategies.

Summarizing this yields the following rate equations:

$$D = \left(1 - \frac{l}{m}\right) \tag{1}$$

$$r_i^{\text{in}} = D \times \left(s_i + \sum_{j=1}^{Q} T_{ji} P_{ji} R_{ji}\right) \tag{2}$$

$$r_i^{\text{out}} = \sum_{i=1}^{Q} T_{ij} P_{ij} R_{ij} \tag{3}$$

where $r_i^{\text{in}}$ and $r_i^{\text{out}}$ are the normalized input and output rates of the queue, respectively, and $Q$ is the total number of nodes in the graph.

Note that $D$ has been defined as a separate value to support the changes necessary for the model defined in section 4.

### 2.2 Model Discussion

#### 2.2.1 Advantages

The leaky bucket strategy described earlier acts as a backchannel for any given node $N_i$. This is because the reliability smoothly changes with the number of messages in the queue. Since this change is smooth, algorithms that can handle smooth increases and decreases (such as gradient descent) and can use this information to optimize the network as it starts to degrade. This contrasts with a strategy of not dropping any packets until the queue is completely filled. In this case, other nodes have no information about the state of the queue until the link suddenly appears to fail completely due to the queue being full and all packets being suddenly dropped.

In addition, the node can choose to "report" other metrics by artificially dropping packets. For example, if a node is running out of energy, it may choose to reduce its own reliability before it runs out of energy. A gradient descent algorithm may view this as a reason to send in another node, which may have more energy. When the first node finally fails, the second node will already be in place to take up position. This same trick can be extended to other metrics.

#### 2.2.2 Disadvantages

The `Useful Model` also has some disadvantages. The first and most obvious is that it is unable to cope with latency requirements. This is because the model does not include time as a variable, which is necessary to determine how close a message is to being past its deadline.

It also does not encode any information about the environment. Without this information, designers are forced to make assumptions about the universe. The simplest assumption is that the universe has no obstacles in it and that the robots can travel in any direction equally well. In short, designers assume that the universe is convex. This is clearly false for the real world. The simplest method to solve this is to extract environmental information from other data models, e.g., a map server. However, if the messages that update the map server and the `Useful Model` become desynchronized, then controllers that depend on the information being coherent may fail. See section 5.2 for more on this.

In addition, the `Useful Model` does not directly incorporate history into itself. The reason that historical information is of interest is simple. As the robots move, they need to know if their movements are actually improving the network, or if the network is degrading, despite what the model claims should be happening. While the fix appears to be simply a matter of maintaining a history buffer of old states, it is actually slightly complicated by the fact that the topology information itself may be old, e.g., in figure 2 consider $N_5$'s knowledge of the links leading to $N_6$. If the topology information is only distributed when messages are sent from node to node, and it has been a long time since messages have been exchanged between $N_5$ and $N_6$, the topology information associated with the links in $N_5$'s communications graph may be significantly out of date. In this case, it may be necessary to flood the network to see if a route still exists and if it has changed significantly. When this flooding should occur, or even if it should occur globally or only within a selected portion of the network, is not addressed by the `Useful Model`.

Finally, and perhaps most subtly, thanks to the fact that all information will be out of date to differing degrees and the fact that the `Useful Model` uses a leaky bucket strategy, the controller that is being used to optimize the network may actually become unstable. This means that the controller will never converge on a global optimum, and in the worst case, may actually drive the robots into worse positions instead of better ones. I do not know of any way to prove a particular solution to the `CMOP` is always stable.

### 2.3 Final Remarks

The `Useful Model` is a superset of the models used in sections 3 and 4. Thus, any limitations of the `Useful Model` automatically transfer over to the models defined in those sections. This is a key point of this report: the controllers that can be developed are always limited by the data model chosen.

# 3. The Model in "Connectivity Maintenance in Mobile Wireless Networks via Constrained Mobility"

In this section, we consider the model and controller defined in reference 4. This is the simplest model and controller that this report considers, and I chose it because it is easiest to understand and implement in general. The goals of this section are not only to understand this model, but also examine how the `Useful Model` in section 2 is a superset of the model used in reference 4. Readers can use the framework outlined for this simple model to understand the more complex models presented in this report.

## 3.1 Model Definition

Reference 4 simplifies the `Useful Model` to consider only the following information:

1. The probability of successful communications between pairs of nodes, $P_{ij}$.
2. For each node $N_i$, the subgraph $G_i$ that includes all nodes within a 2-hop radius of $N_i$. That is, all of $N_i$'s immediate neighbors and all neighbors of $N_i$'s immediate neighbors.

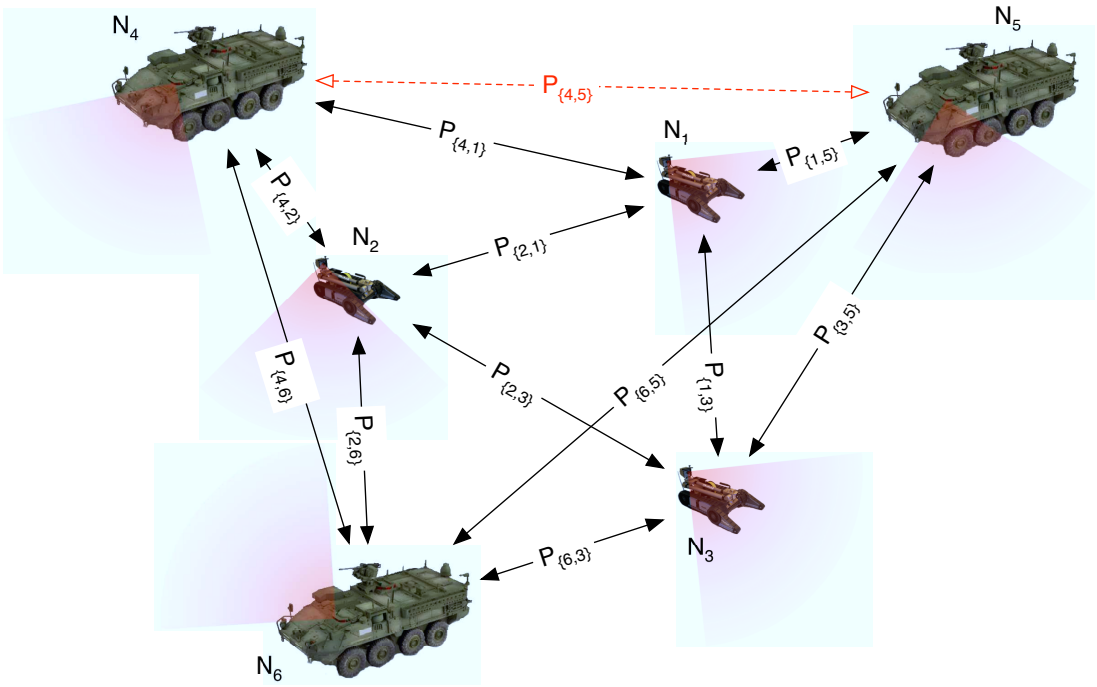
Note that an argument can be made that it also considers the maximum rate of communication across a link $R_{ij}$, but the controller as defined in reference 4 only actually uses the information above.

## 3.2 Controller

The controller defined in reference 4 is the Spreadable Connected Autonomic Network (SCAN) protocol, which simply counts the number of neighbors a node shares with each of its neighbors and stops a node from moving if this falls below a minimum value. That is, for each node $N_i$, whose one-hop neighbors are in the set $\mathrm{Neighbor}(N_i)$, we define the measure

$$k_i = \min_{n \in \mathrm{Neighbor}(N_i)} |\mathrm{Neighbor}(N_i) \cap \mathrm{Neighbor}(n)| \tag{4}$$

If $k_i < C$ for some constant $C$, then node $N_i$ is not permitted to move, otherwise, it is permitted to move in any direction. This is illustrated in figure 3.




| Node | Can move with red link | Can move without red link |
|---|---|---|
| $N_1$ | Yes | No |
| $N_2$ | Yes | Yes |
| $N_3$ | Yes | Yes |
| $N_4$ | Yes | No |
| $N_5$ | Yes | No |
| $N_6$ | Yes | No |

Figure 3. The network model used in reference 4. If all nodes are required to share two neighbors with each of their neighbors ($C = 2$ in equation 4), then while the red dashed link exists, all nodes may move. Without it, only nodes $N_2$ and $N_3$ may move.

To ensure that assessment of the network is done correctly, time is broken up into two types of periods, one for network assessment and one for communications. All nodes calculate equation 4 during the assessment period and communicate during the other period. The periods alternate between each other continuously.

Note that the word "connected" under the SCAN protocol can be interpreted in many different ways. In reference 4, "connected" means that the RSSI measurement is greater than some minimum. However, to translate it to use the information in the graph in section 2, it must use $P_{ij}$ instead. This has no effect on the protocol.

### 3.3 Analysis and Discussion

The SCAN protocol can be summed up as "First, do no harm." The goal of the SCAN protocol is to ensure that if a network is currently connected, then it is impossible for any robot to move to a disconnected position. However, for this to always be true, certain assumptions must be made. These include the following:

- Equation 4 is a good measure of how robust the network is against failures (the assumption is that the greater $k$ is, the less likely a robot will become disconnected).
- Link failures are independent of one another, and no more than $C$ links will have failed during any assessment period. Note that if a robot fails, many links may fail at one time.
- The environment is static. If robots can be forcibly separated by environmental changes (e.g. by a door closing or traveling on an elevator), then the SCAN protocol cannot guarantee connectivity.
- The 2-hop graph can be calculated and distributed quickly enough. If it cannot, then it is possible for a robot to believe that it has satisfied equation 4, while in reality it has broken all connections and has therefore partitioned the network.

These assumptions are not always true, which means that a careful analysis of what happens when one or more assumptions are false needs to be done. Loosely following the logic in reference 4, we define $p_{ij}$ as the probability that the link connecting nodes $N_i$ and $N_j$ fail from one assessment period to the next, and $\tau$ is the average fraction of time for any link that has failed to be down[9]. For any node $N_i$ to lose connectivity with the network, all of its links must fail simultaneously. This can happen with probability $\prod_j p_{ij}$. The fraction of time that $N_i$ is disconnected from the network can then easily be calculated as $1 - \tau \prod_j p_{ij}$. Note that the products are taken over all of the neighbors that node $N_i$ has.

[9]That is, $\tau$ is in the real interval $[0, 1]$, with 0 means that when a link fails, the fraction of time it is down for is no time, and 1 meaning that when a link is down, it down for the whole time period over which it is measured.

We can simplify this significantly if we assume that the probability of failure for any link is the same as for any other link. This results in the simplification $1 - \tau p^{m_i}$, where $m_i$ is the number of neighbors that node $N_i$ has. However, this is not the worst-case scenario. Since $p \leq 1$ is always true, then if $m_i > m_j$, and if $p < 1$, then $p^{m_i} < p^{m_j}$. So small $m$ leads to worse reliability. Nevertheless, there is a minimum value for $m$. As long as the SCAN protocol is obeyed, the minimum value for $m$ is $k$. Thus, to partition the network, we need to break $k + 1$ or more links. This allows us to correlate $k$ with the expected fraction of time that the network is fully connected:

$$1 - \tau p^{k+1} \tag{5}$$

By choosing the fraction of time we require the network to be fully connected, we can calculate a reasonable value for $k$.

### 3.4 Model Discussion

#### 3.4.1 Advantages

As alluded to earlier, one of the advantages of the SCAN protocol is its simplicity. Once equation 4 is understood, it is extremely easy to implement.

Another advantage is its generality. It makes no assumptions about the linearity or convexity of the problem, which makes it trivial to solve the "convex environment" assumption that is present in the `Useful Model` (see section 2.2.2).

#### 3.4.2 Disadvantages

Unfortunately, the SCAN protocol also has a number of disadvantages. If $k$ is too large and a link fails, it may mean that no node is permitted to move. In such a case, the network will be in an unrecoverable state, unless the failed link suddenly starts working again.

In addition, the SCAN protocol is unable to optimize the positions or trajectories of the robots to improve network performance. Provided the current topology meets the minimal constraints, it is always accepted.

Finally, as mentioned in earlier sections, any limitations of the `Useful Model` carry over to the other models, and consequently, controllers that are derived from the `Useful Model`. The SCAN protocol is no exception. Simply stopping may not be the best strategy when a robot has lost some or all connectivity. For example, if the robot has lost connectivity because it has entered a stream or other body of water, it is likely that its communications will fail. Stopping in

the body of water is unlikely to improve communications, but backtracking might. Unfortunately, with the information in the `Useful Model`, it is not possible to decide if backtracking will help or not (because it lacks both historic and environmental information). Thus, the model itself limits what can be done to improve the SCAN protocol.

### 3.4.3 The `CAP` Theorem

As stated in section 1.3.5, designers often choose one of two methods to solve the problems enforced by the `CAP` theorem, by either making assumptions that make the problems go away or relaxing the requirements in Definition 1 and accepting that they cannot guarantee that all nodes will always be connected. Reference 4 takes the second route, stating that it cannot meet all the requirements of the `CAP` theorem. Overall, this is the only correct approach to take. Assuming that one of the properties of the `CAP` theorem is guaranteed and working from there will only result in failure when that assumption proves false.

## 3.5 Final Remarks

The SCAN protocol's advantage of generality combined with its disadvantage of not being able to optimize the network suggests that it should be used as a backstop to other strategies, preventing them from causing the network to fail. This is especially true if it is combined with controllers that are either unaware of or choose to ignore the `CAP` theorem. In those cases, it can be used to address shortcomings of the "more powerful" methods.

---

# 4. The Model in "Distributed Control of Mobility and Routing in Networks of Robots"

---

In this section, we explore the model and controller described in reference 3. Once we show how it is a specialization of the `Useful Model`, we discuss how to derive a controller that uses it.

**Warning:** because of space limitations in this report, this section deliberately glosses over some of the points raised in reference 3. If the reader wishes to implement the algorithm described in reference 3, then I strongly advise read the preprint at http://www.seas.upenn.edu/∼aribeiro/preprints/2012_zavlanos_etal.pdf, which has nuances that were left out of the published version (*3*).

### 4.1 Model Definition

Figure 4 shows a specialized version of the model described in figure 2 taken from reference 3. In this model, there is a difference between robots ($N_1$, $N_2$, $N_3$) and access points ($N_4$, $N_5$, $N_6$)[10]. Access points act as data sinks only; they can never put messages on the network. Additionally, all robots are assumed to have the same maximum message rate ($R_{ij}$ in section 2.1), which is why that value is dropped in this model. Finally, unlike the model in section 2.1, the model in reference 3 does not define what the size of the queue is nor does it define a policy for how to react to a full queue. Instead, the goal of the paper is to develop a controller that ensures that $r_i^{\text{out}} \geq r_i^{\text{in}}$ is always true, which guarantees that all messages are delivered.

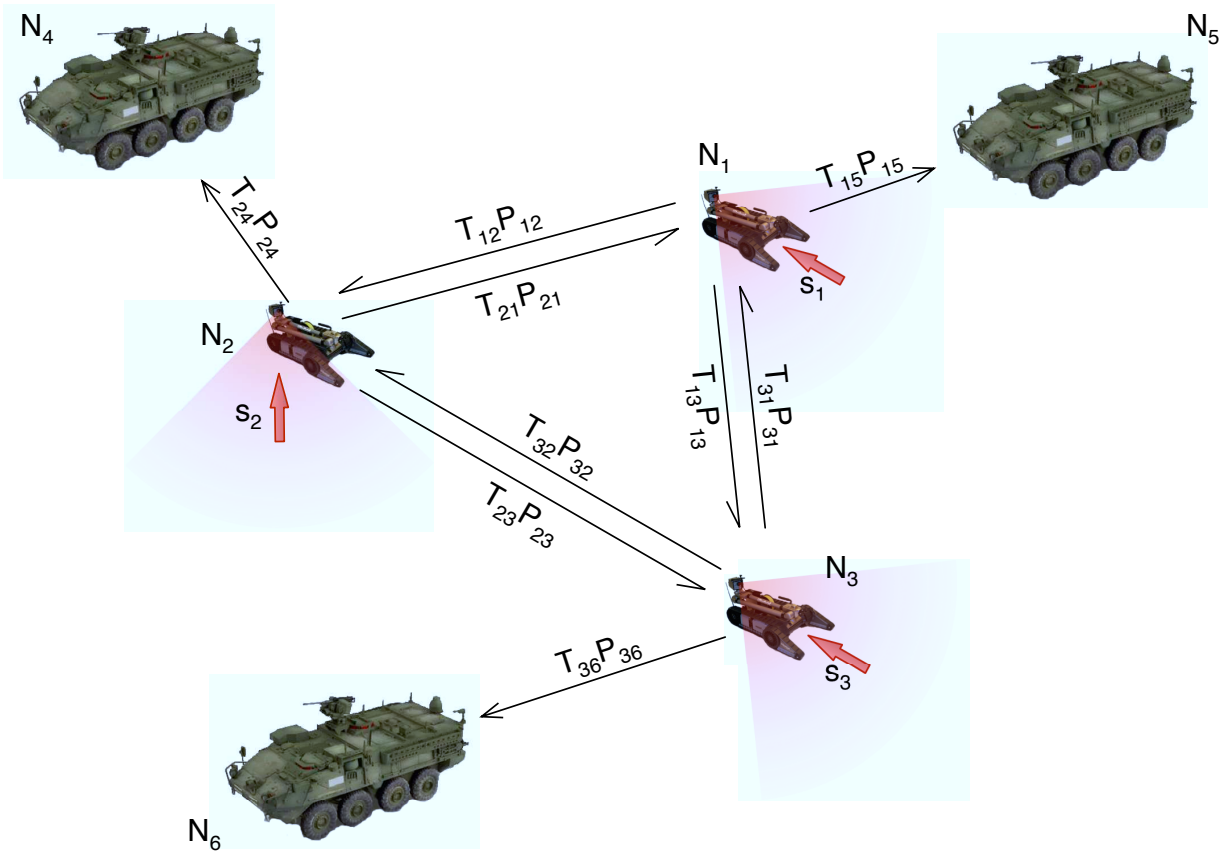



Figure 4. The network model used in (*1*).

[10]Variable names have been changed from what are used in reference 3 to match the names used in this report.

Because of these simplifications from the `Useful Model`, we start to diverge in what and how we define various variables. To start with, unlike in section 2.1 where $Q$ is the total number of nodes in the system, reference 3 defines two variables, $J$ and $K$, which are the number of nodes and the number of access points, respectively. It also sorts ID numbers so that the ID number of any node is less than the ID number of any access point. Finally, unlike in section 2.1, the controller in reference 3 ensures that $r_i^{\text{out}} \geq r_i^{\text{in}}$, which means that $D$ in equation 1 can be set to 1. Section 4.2 explains this further.

Using these definitions, reference 3 defines $r_i^{\text{in}}$ and $r_i^{\text{out}}$ as follows:

$$r_i^{\text{in}} = s_i + \sum_{j=1}^{J} T_{ji} P_{ji} \tag{6}$$

$$r_i^{\text{out}} = \sum_{j=1}^{J+K} T_{ij} P_{ij} \tag{7}$$

This is not the end, however. In section 2, we assumed that the probability distributions could be different depending on the pair of robots under discussion. Reference 1 simplifies this and assumes that there is a single probability distribution that is only a function of the positions of pairs of robots; thus, all robots use the same probability distribution. This leads to the following redefinition:

$$P_{ij} \rightarrow P(\mathbf{x}_i, \mathbf{x}_j)$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the position vectors of the $i$th and $j$th robots, respectively. This leads to the definition of $r_i^{\text{in}}$ and $r_i^{\text{out}}$ that is actually used in reference 3:

$$r_i^{\text{in}} = s_i + \sum_{j=1}^{J} T_{ji} P(\mathbf{x}_j, \mathbf{x}_i) \tag{8}$$

$$r_i^{\text{out}} = \sum_{j=1}^{J+K} T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) \tag{9}$$

### 4.2 Controller

As mentioned in section 4.1, one of the goals of the controller defined in reference 3 is to ensure that for every node $r_i^{\text{out}} \geq r_i^{\text{in}}$. This leads to the following equation:

$$s_i \leq \sum_{j=1}^{J+K} T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) - \sum_{j=1}^{J} T_{ji} P(\mathbf{x}_j, \mathbf{x}_i) \tag{10}$$

If equation 10 is true at all times, then our network is always connected. So let us consider what we have control over. $T_{ij}$ and $T_{ji}$ are the probabilities that a particular link will be chosen to send a message across. We have complete control over this (remember, this is not the probability that a message will be received correctly, it is just the probability that we will use that link to send a message). We can control the robot's position $\mathbf{x}$, but only within the limits imposed by time (how fast they can move) and space (obstacles that are in the way). We have no control over the incoming message rate $s_i$.

This forces us to make some assumptions when designing a controller. The simplest assumption is that the network must guarantee that every node will support some minimal rate $s_{i_\emptyset}$. That is,

$$\forall i, \quad s_{i_\emptyset} \leq s_i \leq \sum_{j=1}^{J+K} T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) - \sum_{j=1}^{J} T_{ji} P(\mathbf{x}_j, \mathbf{x}_i) \tag{11}$$

So, at this point, our goal is to manipulate $T_{ij}$, $T_{ji}$, $\mathbf{x}_i$, and $\mathbf{x}_j$ to guarantee that equation 11 is always true. This is a minimal requirement, though. While a network may function when $\forall i, \ s_i = s_{i_\emptyset}$, it may not be a particularly "good" network. If nothing else, it is not likely to be a robust network because a slight change may cause $s_i > s_{i_\emptyset}$ for some $i$. Thus, we would like a buffer to protect against momentary lapses. A method to do this is to find some optimal operating point, which should provide a buffer, while at the same time making the network better than a minimally acceptable network.

Unfortunately, at the current time, there are no known efficient methods to find the global maximum of an arbitrary function. There are, however, methods of solving convex problems. So, reference 3 defines a pair of utility functions $U_i(s_i)$ and $V_{ij}(T_{ij})$ that are required to be strictly convex, that are used to recast the original problem as a convex optimization problem. Two possible definitions for $U_i(s_i)$ are

$$U_i(s_i) = w_i s_i \tag{12}$$

$$U_i(s_i) = w_i \log(s_i) \tag{13}$$

and two possible definitions for $V_{ij}(T_{ij})$ are

$$V_{ij}(T_{ij}) = 0 \tag{14}$$

$$V_{ij}(T_{ij}) = -w_{ij} T_{ij}^2 \tag{15}$$

These are not the only possible definitions for $U_i(s_i)$ and $V_{ij}(T_{ij})$; the only real requirement is that they both be strictly convex.

Finally, we come to the problem that we are trying to optimize

$$\Psi_{\mathbf{x}} = \max_{s_i, T_{ij}} \sum_{i=1}^{J} U_i(s_i) + \sum_{i=1}^{J} \sum_{j=1}^{J+K} V_{ij}(T_{ij}) \tag{16}$$

with the following constraints $\forall i \in \{1 \dots J\}$:

$$s_i + \sum_{j=1}^{J} T_{ji} P(\mathbf{x}_j, \mathbf{x}_i) \leq \sum_{j=1}^{J+K} T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) \tag{17}$$

$$s_{i_\emptyset} \leq s_i \tag{18}$$

$$\sum_{j=1}^{J+K} T_{ij} \leq 1 \tag{19}$$

At this point, we can start analyzing equation 16. The first thing to notice is that if $U_i(s_i)$ and $V_{ij}(T_{ij})$ are strictly convex, then so is equation 16. That means that any convex optimization techniques can be used here to solve the problem. The second thing to notice is that solving equation 16 requires global knowledge of the network's current topology. This implies a centralized planner. The authors of reference 3 noticed this and developed a better way via the Lagrangian dual, described in section 4.2.1.

### 4.2.1 The Lagrangian Dual and Distributed Optimization

One method of solving a convex problem that has constraints is to solve the Lagrangian dual of the problem. Following the steps described in reference 3, equations 16–17, become

$$\begin{aligned} \mathcal{L}_{\mathbf{x}}(\lambda, \mathbf{T}, \mathbf{s}) = & \sum_{i=1}^{J} U_i(s_i) + \sum_{i=1}^{J} \sum_{j=1}^{J+K} V_{ij}(T_{ij}) \\ & + \sum_{i=1}^{J} \lambda_i \left[ \sum_{j=1}^{J+K} T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) - \sum_{j=1}^{J} T_{ji} P(\mathbf{x}_j, \mathbf{x}_i) - s_i \right] \end{aligned} \tag{20}$$

and with the dual function defined as

$$g_{\mathbf{x}}(\lambda) = \max_{\substack{s_{i_\emptyset} \leq s_i, \\ \sum_{j=1}^{J+K} T_{ij} \leq 1}} \mathcal{L}_{\mathbf{x}}(\lambda, \mathbf{T}, \mathbf{s}) \tag{21}$$

and the problem in the dual space that we want to solve defined as

$$D_{\mathbf{x}} = \min_{\lambda > 0} g_{\mathbf{x}}(\lambda) \tag{22}$$

As noted in section 4.2, equation 16 is convex. This implies that the optimal solution to equation 22 is the same as equation 16, that is, $\Psi^*_{\mathbf{x}} = D^*_{\mathbf{x}}$. By itself, this is not interesting, though. The interesting thing is how equation 20 can be transformed into a sum of local Lagrangians. This can be seen via the following transformations:

$$\begin{aligned}
\mathcal{L}_{\mathbf{x}}(\lambda, \mathbf{T}, \mathbf{s}) &= \sum_{i=1}^{J} U_i(s_i) + \sum_{i=1}^{J} \sum_{j=1}^{J+K} V_{ij}(T_{ij}) \\
&\quad + \sum_{i=1}^{J} \lambda_i \left[ \sum_{j=1}^{J+K} T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) - \sum_{j=1}^{J} T_{ji} P(\mathbf{x}_j, \mathbf{x}_i) - s_i \right] && (23) \\
&= \sum_{i=1}^{J} \Bigg( U_i(s_i) - \lambda_i s_i + \sum_{j=1}^{J+K} \Big[ V_{ij}(T_{ij}) + \lambda_i T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) \Big] \\
&\quad - \sum_{j=1}^{J} \lambda_i T_{ji} P(\mathbf{x}_j, \mathbf{x}_i) \Bigg) && (24) \\
&= \sum_{i=1}^{J} \Bigg( U_i(s_i) - \lambda_i s_i + \sum_{j=1}^{J} \Big[ V_{ij}(T_{ij}) + \lambda_i T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) \Big] \\
&\quad + \sum_{j=J+1}^{J+K} \Big[ V_{ij}(T_{ij}) + \lambda_i T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) \Big] \\
&\quad - \sum_{j=1}^{J} \lambda_i T_{ji} P(\mathbf{x}_j, \mathbf{x}_i) \Bigg) && (25) \\
&= \sum_{i=1}^{J} \Bigg( U_i(s_i) - \lambda_i s_i + \sum_{j=J+1}^{J+K} \Big[ V_{ij}(T_{ij}) + \lambda_i T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) \Big] \\
&\quad + \sum_{j=1}^{J} \Big[ V_{ij}(T_{ij}) + \lambda_i \big( T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) - T_{ji} P(\mathbf{x}_j, \mathbf{x}_i) \big) \Big] \Bigg) && (26)
\end{aligned}$$

We can make one more simplification, if we notice that the range of both $i$ and $j$ is $[1, J]$, we can reorder the way we do the summations to get the following equivalent equation:

$$\begin{aligned}\mathcal{L}_{\mathbf{x}}(\lambda, \mathbf{T}, \mathbf{s}) &= \sum_{i=1}^{J} \Bigg( U_i(s_i) - \lambda_i s_i + \sum_{j=J+1}^{J+K} \Big[ V_{ij}(T_{ij}) + \lambda_i T_{ij} P(\mathbf{x}_i, \mathbf{x}_j) \Big] \\ &\quad + \sum_{j=1}^{J} \Big[ V_{ij}(T_{ij}) + T_{ij} P(\mathbf{x}_i, \mathbf{x}_j)\big(\lambda_i - \lambda_j\big) \Big] \Bigg) \qquad (27) \\ &= \sum_{i=1}^{J} \mathcal{L}_{\mathbf{x},i}(\lambda, \mathbf{T}, \mathbf{s}) \qquad (28)\end{aligned}$$

Notice what we get with this: all of the primal variables associated with node $N_i$ are now associated with the local Lagrangian $\mathcal{L}_{\mathbf{x},i}(\lambda, \mathbf{T}, \mathbf{s})$. Keep this in mind and go back to equations 21 and 22, which are what we really want to solve. Solving those equations means we need to know the primal variables $\mathbf{T}$ and $\mathbf{s}$ that maximize equation 21, and the dual variable $\lambda$ that minimizes equation 22. In short, we want the following:

$$\{s_{\mathbf{x},i}(\lambda)\}_{\forall i}, \{T_{\mathbf{x},ij}(\lambda)\}_{\forall i,j} \triangleq \underset{\substack{s_{i_\emptyset} \le s_i, \\ \sum_{j=1}^{J+K} T_{ij} \le 1}}{\text{argmax}} \quad \mathcal{L}_{\mathbf{x}}(\lambda, \mathbf{T}, \mathbf{s}) \qquad (29)$$

However, since the dual problem is convex and separable (as shown in equation 28), this is the same as finding the primal variables that maximize each of the local Lagrangians. In short,

$$s_{\mathbf{x},i}(\lambda), \{T_{\mathbf{x},ij}(\lambda)\}_{j=1}^{J+K} \triangleq \underset{\substack{s_{i_\emptyset} \le s_i, \\ \sum_{j=1}^{J+K} T_{ij} \le 1}}{\text{argmax}} \quad \mathcal{L}_{\mathbf{x},i}(\lambda, \mathbf{T}, \mathbf{s}) \qquad (30)$$

This is not sufficient, though; $s_{\mathbf{x},i}(\lambda)$ and $\{T_{\mathbf{x},ij}(\lambda)\}_{j=1}^{J+K}$ will only be optimal if $\lambda$ is optimal. Since we do not know if $\lambda$ is optimal, we may have a problem. Fortunately, since the dual problem is convex, it is possible to "walk" towards the global optimum via gradient descent. By using the gradient of equation 21,

$$[\nabla g_{\mathbf{x}}(\lambda)]_i = \sum_{j=1}^{J+K} T_{\mathbf{x},ij}(\lambda) P(\mathbf{x}_i, \mathbf{x}_j) - \sum_{j=1}^{J} T_{\mathbf{x},ij}(\lambda) P(\mathbf{x}_j, \mathbf{x}_i) - s_i(\lambda) \qquad (31)$$

we can update $\lambda$, which can then be used to update $s_{\mathbf{x},i}(\lambda)$ and $\{T_{\mathbf{x},ij}(\lambda)\}_{j=1}^{J+K}$. This iterative loop will eventually converge on the global optimum.

Showing how $\lambda$ is updated requires a dummy index variable. Copying reference 3, we use the dummy index variable $t_n$[11]. Letting $s_i(t_n) = s_{\mathbf{x}(t_n),i}(\lambda(t_n))$ and $T_{ij}(t_n) = T_{\mathbf{x}(t_n),ij}(\lambda(t_n))$, we can rewrite equation 30 as follows (equation 12 in reference 3):

$$s_i(t_n), \{T_{ij}(t_n)\}_{j=1}^{J+K} \triangleq \underset{\substack{s_{i_\emptyset} \leq s_i, \\ \sum_{j=1}^{J+K} T_{ij} \leq 1}}{\text{argmax}} \mathcal{L}_{\mathbf{x}(t_n),i}(\lambda(t_n), \mathbf{T}, \mathbf{s}) \tag{32}$$

and copying equation 13 in reference 3 for the update equation of $\lambda$

$$\lambda_i(t_{n+1}) = \mathbb{P}\Bigg[\lambda_i(t_n) - \epsilon\Bigg(\sum_{j=1}^{J+K} T_{ij}(t_n)P(\mathbf{x}_i(t_n), \mathbf{x}_j(t_n)) - \sum_{j=1}^{J} T_{ij}(t_n)P(\mathbf{x}_j(t_n), \mathbf{x}_i(t_n)) - s_i(t_n)\Bigg)\Bigg] \tag{33}$$

where $\epsilon$ is the usual small positive step value and $\mathbb{P}$ is a projection operator that projects its contents onto the positive orthant.

At this point, we have all the necessary machinery to calculate the gradient in a distributed manner. Note that there is one more subtlety involved here. To calculate equations 32 and 33 requires that each node know its neighbors values for $\lambda_i(t_n)$ and $\{T_{ij}(t_n)\}_{j=1}^{J+K}$; if it is not possible to transmit these values across the network, then it will not be possible to perform the necessary updates and the controller in section 4.2.2 will not work.

### 4.2.2 Control signals

If we followed the gradient defined in equation 31, all robots would converge on a global maximum. However, equation 31 does not have a method of inputting control signals to any of the robots. This means that if we follow the gradient directly, each robots will converge to its own optimal location and then not move from that location.

To solve this, we need a way of modifying a robot's gradient so that the task that we want it to complete has some say in what direction the robot goes. Reference 1 solves this by defining an artificial task potential $\gamma_i$ for each robot $N_i$ that is used in the artificial potential function $\phi_i$.

$$\phi_i = \frac{\gamma_i}{\left(\gamma_i^k + \left((\nabla g_{\mathbf{x},i}(\lambda_i(t_n))) + e\right)^2\right)^{1/k}} \tag{34}$$

[11] $t_n$ is described as discrete time steps in reference 3, but it is not actually time. It is just a dummy index variable that allows us to track from one iteration to another.

where $e$ is a small tolerance term included to ensure that there is a small buffer to protect against random fluctuations that would cause the invariants to become false. Note that equation 34 is similar to, but not the same as, equation 22 in reference 3.

With this artificial potential function, we can control the robots via the following equation:

$$\dot{\mathbf{x}}_i\ (t) = -\alpha \nabla_{\mathbf{x}_i} \phi_i(t) \tag{35}$$

where $\alpha$ is a positive gain to control how fast the robot moves. Note that this is similar to, but not the same as, equation 24 in reference 3.

At this point, we have a complete control loop. Equation 32 gives an estimate of the primal variables, while equation 33 yields an estimate of the dual variable and equation 35 gives the control signal. This is sufficient to control the robot, with the caveats discussed in section 4.3.

### 4.3 Analysis and Discussion

#### 4.3.1 Advantages

The controller in reference 3 is similar to the SCAN protocol in that as the ability of robots to communicate with one another decreases, their ability to move also reduces. Also like the SCAN protocol, because of how equation 34 is defined, it is impossible for the robots to drive themselves to a point where they partition the network, provided the data model is always accurate.

The controller is better than the SCAN protocol in one way, though: robots that obey the control law in equation 35 will always tend to move towards points that optimize the network's reliability. This is unlike the SCAN protocol, which can only prevent disconnection, but has no concept of optimality.

#### 4.3.2 Disadvantages

Unfortunately, the controller suffers from the same problem that all gradient descent methods suffer: if the problem is not convex, then there may be local minima that can trap a solver. In this case, the local minima are not in equation 16, but in the environment. Reference 1 implicitly assumes that the environment is either convex or can be decomposed into a convex free space. Since this is not guaranteed, it is always possible that a robot can become trapped by an obstacle while traveling to its optimal location.

Also, just as limitations of the `Useful Model` affect the SCAN protocol (see section 3.4.2), those same limitations affect the controller in reference 3. Without incorporating information about what the environment is like, the controller is forced to assume that space is convex, and without historical or planning information, the controller can only focus on the quality of the connection at the current moment. This is not necessarily the best way to optimize the network.

In addition, because access points and robots have different characteristics (one can transmit, while the other cannot), the model is slightly more complicated than it needs to be. The only possible reason for doing would be to ensure that access points are not required to move for the system to work. However, this is a weak guess as the algorithm does not require that all members of the network be able to move for it to work correctly (mobile nodes will simply move to slightly different but still optimal positions). Ultimately, permitting the access points to act like nodes in the `Useful Model` may have been mathematically easier, and more useful as well.

Finally, the controller in reference 3 does not guarantee that the network will be robust. It is quite possible that the optimal configuration involves all robots moving into a straight line, where if any robot fails, then the network is partitioned. This is illustrated in the simulation results of reference 3, figure 6.

### 4.3.3 The `CAP` Theorem

Unlike the SCAN protocol of section 3, reference 3 implicitly makes the assumption that the only reason why network partitions may occur is because of controller failure. In fact, as a part of its initial starting conditions, it assumes that the network is connected. Under these assumptions, the problem of partition-tolerance never occurs, meaning that the difficulties of the `CAP` theorem never apply.

This is primary weakness of this algorithm. In the real world, it is impossible to guarantee that the network is never partitioned and assuming that it will not be dangerous.

## 4.4 Final Remarks

As stated in section 3.5, the SCAN protocol is ideal for ensuring that networks always maintain a robust configuration. By combining the SCAN protocol with the gradient descent algorithm in reference 3, it may be possible to ensure that networks are reasonably optimal while guaranteeing a certain $k$-connectivity (and we hope a certain robustness). Alternatively, reference 3 might be extended to include the number of neighbors as a constraints, which can be optimized over in the Lagrangian dual.

# 5. Extensions and Further Research

As shown in section 2, the `Useful Model` has limitations that are inherent in the model and cannot be removed. Because of this, controllers built on the model, such as the ones discussed in section 2–4 have their own inherent limitations. In this section, we explore possible extensions to the model and show what the extra information can alleviate various problems in the model.

## 5.1 Message Extensions

The first extension to consider involves messages rather than the network. As the `Useful Model` is defined, it is impossible to encode the latency requirements of any particular message. The simplest solution is to add an expiration date and importance factor to each message. The expiration date is a point in time in the future when the message is no longer useful. The importance factor allows users to decide if on-time delivery is critical, somewhat important, or unimportant to a particular message. Using this, the `Useful Model`'s first in, first out (FIFO) queue could be converted into a priority queue, where the closer a message is to expiring, the closer to the front of the queue it is moved. This has two effects it reduces latency and increases reliability. The second effect is because every message in the queue may be dropped uniformly at random at the rate $l/m$ (see section 2.1), but since messages that can expire tend to be moved to the front of the queue, they spend less time in the queue, and therefore, have less of a chance of being dropped.

The next extension is to modify the reliability of individual messages. Instead of dropping packets uniformly at random, we can give each packet an "exposure value" in the interval $(0, 1]$. The more exposed it is (the closer to $1$ it is), the more likely it is to be dropped relative to other packets in the queue. This extension must be crafted with care, though; if every message has a low exposure value, then a naïve implementation will overflow the message queue.

## 5.2 Extensions to the `Useful Model`

The simplest extension we can make to the `Useful Model` is to permit each graph edge to have multiple metrics associated with it. This allows tradeoffs such as power needed versus reliability. This kind of tradeoff analysis can be crucial for small sensors that are expected to survive for long periods of time.

We can then consider historical information. Despite the problems noted in section 2.2.2 with trying to add in historical information to the `Useful Model`, such information can aid controllers that are trying to determine reliable routes. For example, although a route might be reliable when it was most recently measured, if historically it had a high variability in reliability, it might be better to either choose a different route, or command a few extra nodes to "plug the hole" by moving towards the unreliable area in order to give more coverage.

We finally come to the two most difficult extensions: environmental and predictive. Adding predictions to the `Useful Model` is fairly simple; we let time extend into the future and record the predictions. As time progresses and we catch up with the predictions, we can record the actual measurements, which allows a running comparison between actual and predicted values. This can permit controllers to select between multiple, competing predictors fairly trivially.

The hard part is adding in the environment. This is equivalent to combining the configuration space $C_r$ and the communications space $N_c$ of section 1.1 together. The problem is that it is not immediately obvious how this can or should be done. The digraph model of the `Useful Model` does not lend itself to extension into space well, and common methods of representing maps (occupancy grids, quad/oct trees, etc.) do not represent the information in the `Useful Model` well[12]. The current default methods that I have seen roboticists use is to not combine the information in a single data model. I currently do not know of any better method of handling this situation.

### 5.3 Final Remarks

There are some low-hanging fruit that can be gathered via some simple extensions to the `Useful Model`. Adding in support for multiple metrics is almost trivial and should be done immediately. Adding in historical support is slightly more difficult, but is also not hard to do. With both of these extensions, a great deal more information can be extracted by controllers that are attempting to solve the `CMOP`.

[12]There are visibility-based models that can extract an estimate of the probability of connectivity from a grid, but this means that there are two separate data models, one for the map of the world, and one for the extracted connectivity graph. This report is talking about a data model where both pieces of information are unified into a single data model.

# 6. Conclusions

In this report, we described the CMOP and various assumptions that algorithm designers make while trying to solve it. We then described the Useful Model, showing how the information in simplified versions of the model is used in two very different algorithms. We also showed how despite the differences in the algorithms, they both suffer from similar limitations imposed by the assumptions of the Useful Model.

The goal of this report is to emphasize how the constraints, abstractions, and assumptions we choose to focus on directly affect the model we develop, and that the model constrains the possible set of solutions we develop. Neither the simple model of section 3 nor the sophisticated model of section 4 are able to fully solve the CMOP, but this is not due to the algorithms involved, but instead to the data model that is used. To develop better controllers that are able to solve the CMOP, we need better data models. This is the only way to truly solve the CMOP.

## 7. References


1. Groshong, K. Mars Rover's Wheel Breaks as Winter Beckons. *New Scientist* **March 2006**, http://www.newscientist.com/article/dn8870-mars-rovers-wheel-breaks-as-winter-beckons.html.

2. NASA Jet Propulsion Laboratory (JPL). Spirit Updates, Spirit Remains Silent at Troy, February 17–23, 2011. http://marsrovers.jpl.nasa.gov/mission/status_spiritAll.html#sol2533 (accessed February 2012).

3. Zavlanos, M.; Ribeiro, A.; Pappas, G. Distributed Control of Mobility & Routing in Networks of Robots. In *Signal Processing Advances in Wireless Communications (SPAWC), 2011 IEEE 12th International Workshop on*, IEEE Signal Processing Society: IEEE Signal Processing Society, Piscataway, NJ, June 2011.

4. Reich, J.; Misra, V.; Rubenstein, D.; Zussman, G. Connectivity Maintenance in Mobile Wireless Networks via Constrained Mobility. In *Proceedings of IEEE Infocom*, IEEE: IEEE Copyrights Manager, IEEE Operations Center, Piscataway, NJ, April 2011.

5. Zhang, R.; Rubin, I. Robust Flow Admission Control and Routing for Mobile Ad Hoc Networks. In *Military Communications Conference, 2006. MILCOM 2006. IEEE*, IEEE Communications Society: New York, NY, October 2006.

6. Gilbert, S.; Lynch, N. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. *SIGACT News* **2002**, *33* (2), 51–59.

7. Tanenbaum, A. S.; Wetherall, D. J. *Computer Networks*; 5 ed.; Prentice Hall: Old Tappan, NJ, 2010.

| NO. OF COPIES | ORGANIZATION |
|---|---|
| 1 (PDF ONLY) | DEFENSE TECHNICAL INFORMATION CTR<br>DTIC OCA<br>8725 JOHN J KINGMAN RD<br>STE 0944<br>FORT BELVOIR VA 22060-6218 |
| 1 | US ARMY RESEARCH LAB<br>IMAL HRA<br>2800 POWDER MILL RD<br>ADELPHI MD 20783-1197 |
| 1 | US ARMY RESEARCH LAB<br>RDRL CIO LL<br>2800 POWDER MILL RD<br>ADELPHI MD 20783-1197 |
| 1 | US ARMY RESEARCH LAB<br>RDRL CII A C KARAN<br>2800 POWDER MILL RD<br>ADELPHI MD 20783-1197 |
| 1 | US ARMY RESEARCH LAB<br>RDRL CII A S YOUNG<br>2800 POWDER MILL RD<br>ADELPHI MD 20783-1197 |